

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)			2. REPORT DATE 18-March-2004	3. REPORT TYPE AND DATES COVERED Final Report March 2000 – November 2003
4. TITLE AND SUBTITLE Language-Based Security for Extensible Systems			5. FUNDING NUMBERS F49620-00-1-0198	
6. AUTHOR(S) Professor Fred B. Schneider and Professor J. Gregory Morrisett				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Cornell University 4130 Upson Hall Ithaca, NY 14853			8. PERFORMING ORGANIZATION REPORT NUMBER 36641	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM 4105 Wilson Blvd. Room 713 Arlington, VA 22203-1954			10. SPONSORING / MONITORING AGENCY REPORT NUMBER N/A	
11. SUPPLEMENTARY NOTES N/A				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public Release; distribution is Unlimited				12b. DISTRIBUTION CODE N/A
13. ABSTRACT (Maximum 200 Words) Successful attacks on computing infrastructures often involve failures of type safety. A major contribution of this grant has been the creation of type systems and type-checking algorithms for low-level languages in use today. In addition, "certifying compilation" was developed to eliminate the need to trust correctness of high-level language implementations. However, ensuring type safety is not sufficient for ruling-out misbehavior in code. A second contribution of this grant was to design and build program-rewriting tools employed for security policy enforcement and also to derive a theoretical characterization for what kinds of policies can be enforced by program rewriting. The theoretical work compares the expressive power of rewriting against traditional security enforcement mechanisms; rewriting is proved to be strictly more powerful. The in-lined reference monitor toolkits handle x86 machine code, the Java virtual machine, and Microsoft's .NET framework.				
14. SUBJECT TERMS Language-based security, in-lined reference monitors, typed assembly language, program rewriting, type systems, proof-carrying code, certifying compilation				15. NUMBER OF PAGES 18
				16. PRICE CODE N/A
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Language-Based Security for Extensible Systems

AFOSR Grant F49620-00-1-0198
Final Report

Fred B. Schneider and Greg Morrisett
Computer Science Department
Cornell University
Ithaca, New York 14853

Objectives

To better support flexibility, evolution, and performance, a new class of system architectures is emerging. Integral to these newer architectures is support for clients to extend service interfaces dynamically. Specifically, clients can send code extensions—perhaps over a network—to services, and these services execute this *foreign code* on behalf of the client.

Unfortunately, the flexibility provided by extensible architectures is also a source of vulnerability, as misbehaved extensions can cause considerable damage. Extensible systems therefore must have security mechanisms to protect against malicious actions by foreign code.

We need mechanisms that support enforcement of a rich class of security policies for extensible systems. The mechanisms should have modest runtime overhead or else a primary attraction of extensible architectures will be lost. And the tension between flexibility and performance is what makes this security problem a particularly difficult one to tackle.

This project studied how programming language technology could be leveraged to support enforcement of rich classes of security policies. We identified and exploited ideas from programming language design, type and

proof systems, semantics, and implementation that provide a basis for meeting the twin goals of flexibility and performance.

Status of Effort

The research supported by this grant developed two major areas of language-based security: (1) Type systems for verifying the memory safety of systems code, and (2) Flexible enforcement of security policies through program rewriting.

Many of the successful attacks on our computing infrastructure involve a failure of type safety. To date, most type-safety analyzers have depended upon the code being written in a high-level, structured language (e.g., Java). But the vast majority of our computing infrastructure is coded in low-level languages such as C. One of our major contribution was the development of type systems and type-checking algorithms for such low-level languages. In addition, we developed a technique called *certifying compilation* that eliminates the need to trust that a high-level language's implementation is correct.

Ensuring type safety is necessary but insufficient to rule out misbehavior in code. We therefore explored a new approach to enforcing desirable behavior based on rewriting executable code. The essence of the approach is that one can express a high-level policy in a declarative language and our rewriting tool in-lines a reference monitor into the binary which enforces that policy. Our research in this area included fundamental theoretical results as well as practical tools. In particular, we developed theoretical models that let us compare the expressive power of rewriting when compared to traditional security enforcement mechanisms and showed that rewriting is strictly more powerful. We also developed in-lined reference monitor toolkits for machine code, the Java virtual machine, and Microsoft's .NET framework.

Accomplishments/New Findings

Our research concentrated on the following areas of language-based security:

- Type systems for verifying the safety of low-level code.
- Inlined reference monitors for enforcing security policies.

Type Systems for Low-Level Code

Today, our computing and communications infrastructure is built using unsafe, error-prone languages such as C or C++ where buffer overruns, format string errors, and space leaks are not only possible, but frighteningly common. In contrast, type-safe languages, such as Java, Scheme, and ML, ensure that such errors either cannot happen (through static type-checking and automatic memory management) or at least are caught at the point of failure (through dynamic type and bound checks). This fail-stop guarantee is not a total solution, but it does isolate the effects of failures, facilitates testing and determination of the true source of failures, and it enables tools and methodologies for achieving greater levels of assurance.

The obvious question is: “Why don’t we re-code our infrastructure using type-safe languages?” Though such a technical solution looks good on paper, the cost is simply too large. For instance, today’s operating systems consist of tens of millions of lines of code. Throwing away all of that C code and reimplementing it in, say Java, is simply too expensive.

Under the auspices of this grant, we have explored how to adapt type systems to low-level languages, such as C/C++ and even assembly language. The goal has been to (a) provide effective tools that allow current systems to be statically or dynamically checked to ensure type safety, and (b) to eliminate the need to trust those tools through the process of *certifying compilation*.

Cyclone Compiler

As a part of this research, we developed Cyclone, a type-safe extension to the C programming language. The type system of Cyclone accepts many C functions without change and uses the same data representations and calling conventions as C. The Cyclone type system also rejects many C programs to ensure safety. For instance, it rejects programs that perform (potentially) unsafe casts, that use unions of incompatible types, that (might) fail to initialize a location before using it, that use certain forms of pointer arithmetic, or that attempt to do certain forms of memory management.

All of the analyses used by Cyclone are local (i.e., intra-procedural) so that we can ensure scalability and separate compilation. The analyses have also been carefully constructed to avoid unsoundness in the presence of threads. The price paid is that programmers must sometimes change type

definitions or prototypes of functions, and occasionally they must rewrite code.

We find that programmers must touch about 10% of the code when porting from C to Cyclone. Most of the changes involve choosing pointer representations and only a very few involve region annotations (around 0.6 % of the total changes). So, we developed a semi-automatic tool that can be used to automate most of these changes.

The performance overhead of the dynamic checks depends upon the application. For systems applications, such as a simple web server, we see no overhead at all. This is not surprising, as these applications tend to be I/O-bound. For scientific applications, we were seeing a much larger overhead (around 5x for a naive port, and 3x with an experienced programmer), due to array bounds and null pointer checks. To avoid these, we incorporated a sophisticated intra-procedural analysis that eliminates most of those checks. For instance, a simple matrix-multiply now runs as fast as C code, where before, it was taking over 5x as long.

We also introduced new typing mechanisms that support a wide range of safe memory management options. Initially, we had to restrict programmers to using only garbage collection, stack allocation, or limited forms of region allocation, all of which could adversely affect time and space requirements. But we have since added support for dynamic region allocation, unique pointers, and reference-counted objects. These mechanisms let programmers control memory management overheads without sacrificing safety. For instance, we were able to improve the throughput of the MediaNet streaming media server by up to 42% and decrease the memory requirements from 8MB to a few kilobytes using these new features.

Cyclone is actively used by the research community to ensure safety for real systems code. For instance, AT&T researchers are using Cyclone to develop a number of high-confidence systems; researchers at Washington and Utah are using Cyclone to develop extensible protocols; and researchers at the Leiden Institute have used Cyclone to develop secure kernel extensions for Linux. They have found Cyclone attractive because the programming model is close to C but provides the strong safety guarantees need for secure systems.

Typed Assembly Language

Type safe languages such as Cyclone can, in principle, provide strong security guarantees. However, the Cyclone compiler and the associated tools are well over 200,000 lines of code. It is likely that there are bugs in these tools which could be exploited by an attacker. Cyclone is not alone in this regard—the type safety of any language (e.g., Java) depends upon the correctness of the implementation of that language, including the compiler or interpreter, the libraries, and the run-time system. These software systems are large and experience has shown that we cannot depend upon them being 100% correct.

The goal of the Typed Assembly Language (TAL) research was to eliminate the need to trust language implementation tools. In particular, the TAL project developed a type system for Intel x86 machine code and a type-checker which consisted of roughly 20,000 lines of code. With the TAL type-checker, it becomes possible to check that a compiler for a high-level language, such as Cyclone or Java, is producing code that actually is type-safe. Once again, we must trust the TAL type-checker, but it is an order of magnitude smaller than the Cyclone compiler (and two orders of magnitude smaller than Sun’s Java implementation) which no longer needs to be trusted.

The primary challenge in developing TAL was finding a set of type constructors that supported compilation of a wide variety of source programming languages. To keep the type system small but flexible, we adapted a suite of higher-order type constructors which could be combined to build higher-level typing abstractions. For instance, TAL had no built-in notion of procedure call and return. Rather, it had simple type constructors for describing machine states at each program point and these type constructors could be composed to specify typing pre- and post-conditions for procedures. This degree of flexibility was crucial for supporting a wide variety of languages.

Inlined Reference Monitors

Inlined reference monitors (IRM) are a new approach to implementing traditional reference monitors. A desired end-to-end security policy is formulated using a high-level declarative policy language, and then a rewriting tool is used to automatically rewrite untrusted code into code that respects the policy. The rewriting tool works by inserting extra state and dynamic checks into the untrusted code so that the code becomes self-monitoring.

Under the period of this funding, our two PSLang/PoET implementations of Java 2 stack inspection were completed and analyzed. We reproduced Wallach’s “security passing style” implementation of the stack inspection policy and obtained comparable performance, and we devised a new implementation of the policy and obtained superior performance. The new implementation works by carefully allocating work so that frequently executed JVM instructions bear relatively less of the burden associated with enforcement. The implementation exhibits performance that is competitive with the JVM-resident stack inspection implementation included in the commercial Java distribution.

We also implemented a prototype deployment of an IRM for a production operating system. Specifically, a set of kernel modifications was developed to support a prototype IRM rewriter in Microsoft’s Windows. This work suggests the need for mechanism to identify which policy is applied to any given executable and for mechanism to manage multiple executables (each enforcing a different policy). For example, .NET caches dll’s (executables), and the architecture for how that cache is managed needs to work differently when the same dll could have been rewritten in multiple ways (to enforce one or another different policies).

In addition, a prototype MSIL (Microsoft Intermediate language) in-lined reference monitor realization was completed. It implements an aspect-oriented programming metaphor for MSIL assembly language (rather than for a high-level language). An aspect-oriented program comprises *aspects*, each of which consists of a *point-cut* and some *advice*. The point-cut is a predicate that specifies where to do rewriting in target code, and the advice specifies how to do the rewriting. Designing a point-cut language that provides complete visibility at a high-level into an assembly language is an interesting challenge. We subsequently extended this prototype so that we could perform arbitrary rewriting on the CIL code by building on a bytecode-rewriting toolkit developed by Microsoft Researchers.

Working with Ph.D. student Kevin Hamlen, we developed a more refined characterization of what policies can be enforced using reference monitors. This new work extends earlier work by Schneider, now taking into account the limits of computability. Specifically, we developed a model based on standard Turing machines, adapted Schneider’s criteria for enforceable security policies, and introduced computability requirements. We also integrated static analysis and program rewriting into the model.

By providing this unifying model, and by basing it on Turing machines,

we were able to compare the relative power of the various enforcement mechanisms, and to relate them to standard computability results. For instance, it was relatively easy to show that the class of policies precisely supported by static analysis could also be supported by both reference monitors and by program rewriting. In addition, we found that introducing a computability requirement on reference monitors was necessary, but not sufficient, for precise characterization of the class of policies actually realizable by reference monitors. And we identified a new property, which we call “benevolence” that provides a more accurate upper bound on the power of reference monitors.

Our most surprising and important results involve program rewriting. We can show that the class of policies originally characterized by Schneider does not include all policies enforceable through rewriting (and vice versa). Indeed, we were able to show that the class of policies enforceable through rewriting does not correspond to any class of the Kleene hierarchy. This is a surprising and important result, as it shows that rewriting truly is a powerful security enforcement technique.

Personnel Supported

Faculty: Greg Morrisett and Fred B. Schneider.

Postdoctoral Researchers: Amal Ahmed, Mike Hicks, Yaron Minsky, Mike Marsh.

Graduate Students: James Cheney, Ulfar Erlingsson, Neal Glew, Daniel Grossman, Kevin Hamlen, Yaron Minsky, Frederick Smith, David Walker, Stephanie Weirich, Steve Zdancewic, and Lidong Zhou.

Publications

1. S. Bellovin, F.B. Schneider, and A. Inouye. Building trustworthy systems: Lessons from the PTN and Internet. *IEEE Internet Computing*, 3, 5 (November-December 1999), 64–72.

2. James Cheney and Christian Urban. System description: Alpha-Prolog, a fresh approach to logic programming modulo alpha-equivalence. *Workshop on Unification* (Valencia, Spain, May, 2003).
3. Karl Crary and Stephanie Weirich. Resource bound certification. *ACM Symposium on Principles of Programming Languages* (Boston, Massachusetts, January 2000), 184–198.
4. Karl Crary, Stephanie Weirich, and Greg Morrisett. Intensional polymorphism in type erasure semantics. *Journal of Functional Programming*, 12(6):567–600, November 2002.
5. Ulfar Erlingsson and Fred B. Schneider. SASI enforcement of security polices: A retrospective. *DARPA Information and Survivability Conference and Exposition (DISCEX'00)* (Hilton Head, South Carolina, January 2000), IEEE Computer Society, Los Alamitos, California, 287–295.
6. Ulfar Erlingsson and Fred B. Schneider. IRM enforcement of Java stack inspection. *Proceedings 2000 IEEE Symposium on Security and Privacy* (Oakland, California, May 2000), IEEE Computer Society, Los Alamitos, California, 246–255.
7. David Gries and Fred B. Schneider. Formalizations of substitutions of equals for equals. *Millennial Perspectives in Computer Science, Proceedings of the 1999 Oxford-Microsoft Symposium in honour of Professor Sir Anthony Hoare*, (Davies, Roscoe, and Woodcock, editors) Palgrave Publishers, Hampshire, England. November 2000, 119–132.
8. Dan Grossman. Existential Types for Imperative Languages. Type checking systems code. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 21–35.
9. Daniel J. Grossman. *Safe Programming at the C Level of Abstraction*. Ph.D. Thesis, Cornell University, August 2003.
10. Daniel J. Grossman. Type-Safe Multithreading in Cyclone. *ACM Workshop on Types in Language Design and Implementation* (New Orleans, LA, January 2003).

11. Dan Grossman and Greg Morrisett. Scalable Certification for Typed Assembly Language. *2000 ACM SIGPLAN Workshop on Types in Compilation*. Lecture Notes in Computer Science, Vol. 2071 (Robert Harper, editor), Springer Verlag, Montreal, 2000, pp. 117-146.
12. D. Grossman, G. Morrisett, T. Jim, M. Hicks, J. Cheney, and Y. Wang. Region-based memory management in Cyclone. *ACM Conference on Programming Language Design and Implementation* (Berlin, Germany, June 2002), 282–293.
13. Dan Grossman, Steve Zdancewic, and Greg Morrisett. Syntactic Type Abstraction. *ACM Transactions on Programming Languages and Systems*. 22(6), November 2000, pp. 1037-1080.
14. Michael Hicks, Stephanie Weirich, and Karl Crary. Safe and flexible dynamic linking of native code. *Proceedings of the ACM Workshop on Types in Compilation* (Montreal, Canada, September 2000), Published as Carnegie Mellon Technical Report CMU-CS-00-161.
15. T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of C. *Usenix Annual Technical Conference* (Monterey, CA, June 2002), 275–288.
16. Dag Johansen, Robbert van Renesse, and Fred B. Schneider. WAIF: Web of Asynchronous Information Filters. *Future Directions in Distributed Computing*, Lecture Notes in Computer Science, Volume 2585 (Schiper, Shvartsman, Weatherspoon, and Zhao, eds.) Springer-Verlag, 2003, 81–86.
17. Gary McGraw and Greg Morrisett. Attacking malicious code: A report to the Infosec Research Council. *IEEE Software* 15, 5 (September/October 2000).
18. G. Morrisett. Type checking systems code. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 1–5.
19. G. Morrisett, K. Crary, N. Glew, and D. Walker. Stack-based typed assembly language. *Journal of Functional Programming* 12, No. 1 (January 2002), University Press, Cambridge, England, 43–88.

20. Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From System-F to typed assembly language. *ACM Transactions on Programming Languages and Systems*, 21(3):528–569, May, 2000.
21. Y. Minsky. *Spreading Rumors Cheaply, Quickly, and Reliably*. Ph.D. Thesis, Cornell University, May 2002.
22. Yaron Minsky and Fred B. Schneider. Tolerating malicious gossip. *Distributed Computing* 16, 1 (Feb 2003), 49–68.
23. Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security* 3, 1 (February 2000), 30–50.
24. Fred B. Schneider. Editorial: Time for Change. *Distributed Computing* Vol. 13, No. 4 (November 2000), 187.
25. Fred B. Schneider. Open source in security: Visiting the bizarre. *Proceedings 2000 IEEE Symposium on Security and Privacy* (Oakland, California, May 2000), IEEE Computer Society, Los Alamitos, California, 126–127.
26. Fred B. Schneider. Least privilege and more. *Computer Systems: Papers for Roger Needham*, Andrew Herbert and Karen Sparck Jones, eds. Microsoft Research, 2003, 209–213.
27. Interview with Fred B. Schneider. *Distributed Systems Online*. <http://www.computer.org/channels/ds>.
28. Fred B. Schneider, Greg Morrisett, and Robert Harper. A language-based approach to security. *Informatics: 10 Years Back, 10 Years Ahead*. Lecture Notes in Computer Science, Vol. 2000 (Reinhard Wilhelm, editor), Springer Verlag, Heidelberg, 2000, pp. 86-101.
29. Frederick Smith. *Certified Run-Time Code Generation*. Ph.D. Thesis, Cornell University, January 2002.
30. Frederick Smith, Dan Grossman, Greg Morrisett, Luke Hornof, and Trevor Jim. Compiling for template-based run-time code generation. *Journal of Functional Programming*, 13(3):677-708, May 2003.
31. Frederick Smith, David Walker, and Greg Morrisett. Alias types. *European Symposium on Programming* (Berlin, Germany, March 2000).

32. David Walker. A type system for expressive security properties. *ACM Symposium on Principles of Programming Languages* (Boston, Massachusetts, January 2000), 254–267.
33. David Walker and Greg Morrisett. Alias types for recursive data structures. *Proceedings of the ACM Workshop on Types in Compilation* (Montreal, Canada, September 2000), Published as Carnegie Mellon Technical Report CMU-CS-00-161.
34. David Walker and Greg Morrisett. Alias Types for Recursive Data Structures. In *2000 ACM SIGPLAN Workshop on Types in Compilation*. Lecture Notes in Computer Science, Vol. 2071 (Robert Harper, editor), Springer Verlag, Montreal, 2000, pp. 177-206.
35. Stephanie Weirich. Type-Safe Cast: Functional Perl. In *Proceedings of the Fifth ACM International Conference on Functional Programming*. Montreal, September 2000, pp. 58-67.
36. David Patrick Walker. *Typed Memory Management*. Ph.D. Thesis, Cornell University, January 2001.
37. Stephanie Weirich. Encoding Intensional Type Analysis. In *European Symposium on Programming*. Lecture Notes in Computer Science, Vol. 2027, Springer Verlag, Genova, Italy, April 2001.
38. Stephanie Weirich. Higher-order intensional type analysis. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 98–114.
39. Stephanie Weirich. *Programming With Types*. Ph.D. Thesis, Cornell University, July 2002.

Interactions/Transitions

Presentations at Meetings, Conferences, Seminars, etc

Invited Lectures: F.B. Schneider

1. Mobile Agent Miracles And Graveyards. Panalist, ASA/MA 99, Palm Springs, California, October 1999.

2. Fault-tolerance Issues For Mobile Agents. Dartmouth Workshop on Transportable Agents, Palm Springs, California, October 1999.
3. Enforceable Security Policies. Qualcom Distinguished Lecture Series, Department of Computer Science, University of California at San Diego, October 1999.
4. Trust in Cyberspace. Federal Communications Commission, Washington, D.C., October 1999.
5. Reinventing Security for Trust in Cyberspace. Banquet speaker. Beyond Moore's Law: Opportunities and Threats from Future, Ubiquitous, High-Performance Computing, Center for Global Security Research, Lawrence Livermore National Laboratory, December 1999.
6. Radio interview, "All Things Considered", National Public Radio, December 17, 1999.
7. Defense Against Malicious Content and Code. ISAT Meeting on Mobility and Security, Software Engineering Institute, Pittsburgh, PA, January 2000.
8. It Depends on What You Pay. Malicious Code Infosec Science and Technology Study Group, San Antonio, Texas, January 2000.
9. SASI Enforcement Of Security Policies: A Retrospective. Information and Survivability Conference and Exposition (DISCEX'00), Hilton Head, South Carolina, January 2000.
10. Containment and Integrity of Mobile Code. Intrusion tolerant systems investigator's meeting, Aspen, Colorado, February 2000.
11. The Non-technical Take on Computing System Trustworthiness. Holy Cross University, Worcester, Mass. March 2000.
12. Secret Sharing Tutorial. University of Tromso, Tromso, Norway. March 2000.
13. AFRL/Cornell Information Assurance Institute. Air Force Rome Laboratories, Rome, New York. April 2000.

14. Overview of Mobile Code Security. Air Force Rome Laboratories, Rome, New York. April 2000.
15. Network Information System Trustworthiness. Short Course on Competitive Strategies for E-Commerce, Johnson Graduate School of Management, Ithaca, New York. April 2000.
16. The Non-technical Take on Computing System Trustworthiness. Jones Seminars on Science, Technology and Society. Dartmouth College, Hanover, N.H. April 1999.
17. Radio interview, “All Things Considered Weekend Edition”, National Public Radio, May 6, 2000.
18. Are There Systems Principles or Only Systems Principals? Invited speaker. In Pursuit of Simplicity. A Symposium Honoring Professor E.W. Dijkstra. Austin, Texas. May 2000.
19. Open Source in Security: Visiting the Bizarre. Panelist. IEEE Symposium on Security and Privacy. Oakland, California. May 2000.
20. In-lined Reference Monitors. InCert Software. Cambridge, Mass. May 2000.
21. The case for language based security. Invited Lecture. Informatics—10 Years Back, 10 Years Ahead. Saarland University, Saarbrucken, Germany. August 2000.
22. In-lined reference monitors. Microsoft Research. Redmond, Washington. October 2000.
23. Radio interview, “All Things Considered”. October 27, 2000.
24. The case for language based security. IFIP wg2.3. Santa Cruz, California. January 2001.
25. The design and deployment of COCA. Distinguished lecture series. SUNY Stony Brook. Stony Brook, New York. February 2001.
26. Fast P2P possibilities. Tysil, Norway. February 2001.

27. The design and deployment of COCA. Department of Computer Science. University of Tromso. Tromso, Norway. February 2001.
28. The case for language based security. Keynote Address, ACM Southeast Conference 2001, Athens, Georgia. March 2001.
29. The design and deployment of COCA. Department of Computer Science. University of Texas, Austin, Texas. March 2001.
30. The case for language based security. IBM Corporation Hawthorne, New York. April 2001.
31. The design and deployment of COCA. AFOSR Principal Investigators Meeting. Ithaca, New York. May 2001.
32. The design and deployment of COCA. Information Assurance Institute Seminar Series. AFRL/IF Rome Research Site, Rome, New York. June 2001.
33. The case for language based security. AFCEA Conference. Hamilton, New York. June 2001.
34. Escaping the ivy tower: Transitioning technology from a university. AFCEA Conference. Hamilton, New York. June 2001.
35. Language-based Security: What's needed and why. Keynote Speaker. Static Analysis Symposium (SAS'01) Paris, France. July 2001.
36. Research overview. Invited panelist, Intel Corporation Microprocessor Research Lab. Santa Clara, California, October 2001.
37. AFRL/Cornell Information Assurance Institute. AFRL Rome Laboratories, Science Advisory Board Review, Rome, New York, November 2001.
38. Mobile code research: Looking back and peering ahead. Keynote Lecture, Fifth IEEE International Conference on Mobile Agents, Atlanta, Georgia, November 2001.
39. Presentation of the International Panel's Review of UK Research in Computer Science. International Review of UK Research in Computer Science—Presentation of the Report and Discussions on the Way Forward, London, England, December 2001.

40. The Case for Language-Based Security. Invited lecture, Intel Research Professor Forum, Santa Clara, California, January 2002.
41. The Case for Language-Based Security. Keynote lecture, Symposium on Cyber Security and Trustworthy Software, Stevens Institute of Technology, Hoboken, New Jersey, March 2002.
42. The Case for Language-Based Security. Department of Computer Science, University of Tromso, Tromso, Norway, March 2002.
43. Research to Support Robust Cyber Defense. High Assurance Systems Workshop, MITRE Corp., Reston, Virginia, May 2002.
44. Asynchronous Proactive Secret Sharing. AFOSR PI Meeting, Syracuse, New York, June 2002.
45. System evaluations. Invited lecture, Intrusion Tolerant Systems Workshop. Washington, D.C. June 2002.
46. Design and Deployment of COCA. Workshop on Dependability and Survivability. IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance. Hilton Head, South Carolina, June 2002.

Invited Lectures: G. Morrisett

1. Language and Compiler Support for Security in Mobile Code. Invited speaker. INFOSEC Working Group on Mobile Code, Washington D.C., October 1999.
2. Type Systems for Low Level Languages. Invited speaker. OpenSIG'99, Pittsburgh, PA, October 1999.
3. Mobile Code Security. Invited speaker. Scientific Advisory Board, Air Force Research Laboratory, Rome, NY, December 1999.
4. The Role of Type Systems in Mobile Code Security. DARPA ISAT Workshop on Mobile Code, Pittsburgh, PA, January 2000.
5. Typed Intermediate Languages. Panelist. Workshop on Foundations of Object Oriented Languages (FOOL), Boston, Mass. January, 2000.

6. Mobile Code Security: An Overview. Invited speaker. TARA Review. Air Force Research Laboratory, Rome, NY, March 2000.
7. Open Issues in Certifying Compilers. Workshop on Proof Carrying Code. Invited Speaker. Santa Barbara, California, June 2000.
8. Open Issues in Proof Carrying Code. InCert Software. Cambridge, Mass. May 2000.
9. Next Generation Low-Level Languages. Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management (SPACE 2001), London, England. January 2001.
10. Towards Next Generation Low-Level Languages. Harvard University, Cambridge, Massachusetts. February 2001.
11. Towards Next Generation Low-Level Languages. Cornell University, Ithaca, New York. February 2001.
12. Next Generation Low-Level Languages. University of Minnesota, Minneapolis, Minnesota. April 2001.
13. Language-Based Security. Danish Technical Institute (ITU), Copenhagen, Denmark, June 2001.
14. Cyclone: A Next-Generation Systems Language. Information Assurance Institute Seminar Series. AFRL/IF Rome Research Site, Rome, New York. August 2001.
15. Next Generation Type Systems. University of Washington and Microsoft Research Summer Institute. Seattle, Washington. August 2001.
16. Explicit Regions in Cyclone. Invited lecture, New England Programming Languages Seminar. Boston, Massachusetts, October 2001.
17. Typed Assembly Language Background. Invited lecture, Intel Research Professor Forum, Santa Clara, California, January 2002.
18. Type Checking Systems Code. Invited lecture, Yale University, New Haven, Connecticut, February 2002.

19. Runtime Code Generation. IFIP Working Group 2.8 on Functional Programming, Las Vegas, Nevada, March 2002.
20. Type Checking Systems Code. Invited lecture, European Symposium on Programming, Grenoble, France, April 2002.
21. Type Checking Systems Code. Invited lecture, Digital, Inc. Washington, D.C. April 2002.

Consultative and Advisory Functions

- Schneider chaired a study for DARPA IPTO Program Manager Jay Lala on promising research directions for Self-Healing Networked Information Systems.
- As a consultant to DARPA/IPTO, Schneider chaired the independent evaluation team for the OASIS Dem/Val prototype project. This project funded two consortia to design a battlespace information system intended to tolerate a class A Red Team attack for 12 hours.
- Microsoft researchers collaborated with Morrisett on the design and implementation of a low-level, type-safe language for building device drivers.
- Greg Morrisett spent nine months visiting Microsoft's Cambridge Research Laboratory, where he worked with researchers on programming language and security technology. In particular, Morrisett worked on the development of Microsoft's tools for automatically finding security flaws in production code, based on his experience with Cyclone. He also worked with student Kevin Hamlen and Microsoft researchers on the implementation of the .NET rewriting tool for inline reference monitors.

Transitions

- Researchers at Carnegie-Mellon University, Princeton University, University of California (Riverside), University of Newcastle-Upon-Tyne, and Intel Research are all now building on PoET/PSLang IRM tools developed by Schneider and collaborators.

- AT&T research collaborated with us to develop the Cyclone language, compiler, and tools. In addition, researchers at the University of Maryland, the University of Utah, Princeton, and the University of Pennsylvania, and Cornell are all using Cyclone to develop research prototypes.
- Researchers at Leiden Institute of Advanced Computer Science in the Netherlands have developed an extension of the Linux operating system, whereby untrusted modules, written in Cyclone, can be dynamically loaded and executed in the context of the kernel.

New discoveries, inventions, or patent disclosures

None.

Honors and Awards

F.B. Schneider:

- Fellow, American Association for Advancement of Science (1992).
- Fellow, Association for Computing Machinery (1994).
- Professor-at-Large, University of Tromso, Tromso, Norway (1996–2004).
- Daniel M. Lazar Excellence in Teaching Award (2000).
- Doctor of Science (honoris causa), University of NewCastle-upon-Tyne (2003).

G. Morrisett:

- Sloan Fellow (1998).
- NSF Faculty Early Career Development (1999).
- Presidential Early Career Award for Scientists and Engineers (2000).
- Allen Newell Medal for Research Excellence, Carnegie Mellon University (2001).
- Ralph Watts Excellence in Teaching Award, Cornell University (2001).
- Allen B. Cutting Chair of Computer Science, Harvard University (2004).